**Module Loaders**

# **How to Create Angular 2 Application Using System.JS**

## What is Webpack

The dynamic web applications usually have lots of javascript files. These files are either created by you or it could be third party libraries like jquery/bootstrap etc. We include these in our **index.html** file using ***<script>*** tag. When a user sends requests to our application, the browser requests and loads these files one at a time. If you have lots of these files, then it will make your application slow. The solution to this problem is to merge all these files into a one or two files so that the browser can download the entire file in one request. This is where **Webpack** is used.

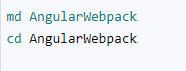
Webpack is a powerful module bundler, which scans your web application looking for javascript files and merges them into one ( or more) big file. Webpack has the ability to bundle any kind of file like JavaScript, CSS, SASS, LESS, images, HTML, & fonts etc. Webpack also comes with Development Server that supports hot module reloading.

## Setting up an Angular Application

The Setting up and angular Application requires the following steps

1. Create an Application folder
2. Create package.json configuration file
3. Installing Dependencies
4. Create the Component
5. Create the root module
6. Bootstrap our application
7. Create the index.html
8. Configure our Application
9. Run the application

## Create an Application Folder

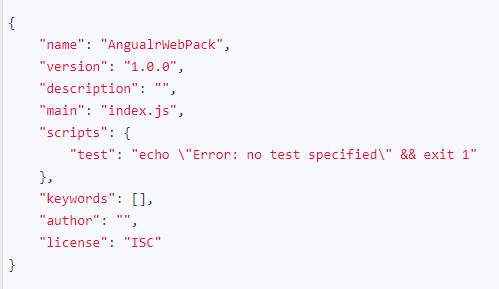


## package.json Configuration file

A **package.json** file contains the metadata about our application. It includes the list of Javascript libraries that are used by our application. The NPM package manager reads this file to install/update the libraries.

You can manually create the package.json file and run the command “npm install” to install the dependencies.





## Installing Dependencies

The next step is to download and install Angular & other libraries. We will use NPM package manager to install them.

#### Installing Angular libraries

Run the following command from the command prompt to install the Angular components.

npm install @angular/common @angular/compiler @angular/core @angular/forms  @angular/http @angular/platform-browser @angular/platform-browser-dynamic  @angular/router --save

The **–save** option ensures that these libraries are saved to package.json file

#### Installing third party libraries

The Angular requires you to install the following dependencies



**Rxjs or Reactive Extensions (Rx)** is a library for composing asynchronous and event-based programs using observable sequences and LINQ-style query operators.

**Zone** is used for change detection

**Core-js** is a ES6 polyfill for typescript.

#### Installing Development Dependencies

The development dependencies are those libraries, which required only to develop the application. For Example, javascript libraries for unit tests, minification, module bundles are required only at the time of development of the application.

Our Angular application needs Typescript. Webpack module, loaders,  plugins, and tools for testing the app etc.

#### Typescript

Typescript is a superset of Javascript. It extends the Javascript and brings options like static typing, classes, interfaces. The Code written in typescript cannot be used directly in the web browser. It must be compiled to javascript before running in the web browser. This process is known as Transpiling. The Transpiling is done by Typescript compiler (tsc)



#### Typings

Typescript forces you to define the types before using them. This has great advantages as any errors are detected at the compile time rather than at the run time.

But our application is going to use a lot of Third Party Javascript libraries in our application. Being written in Javascript, they do not have types defined in them. The typescript compiler will throw an error as it does not recognize them.

This where type definition files come into play. They provide the type information to these javascript libraries



#### Webpack loaders and plugins

Webpack supports custom loaders and plugins. A loader is a program that allows you to preprocess files as you “load” them. They extract the content of the file, transform them and then return the transformed content to Webpack for bundling. With the help of leaders, the Webpack can handle any type of files.

npm install angular2-template-loader awesome-typescript-loader css-loader file-loader html-loader null-loader raw-loader style-loader to-string-loader --save-dev

##### Webpack plugins

A plugin is a program that changes the behaviour of Webpack

npm install html-webpack-plugin webpack-merge extract-text-webpack-plugin --save-dev



npm install jasmine-core karma karma-chrome-launcher karma-jasmine karma-sourcemap-loader karma-webpack --save-dev

create root module,root component,bootstrap,template,css

### Typescript

Create the file **tsconfig.json** in the**src folder** our project and copy the following



### Webpack Bundle

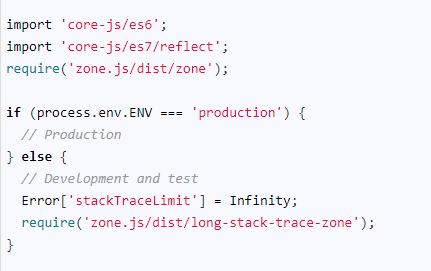
The next step is to configure the Webpack. Webpack allows us to bundle all our javascript files into a one or more files. Let us create three bundles in our application

In the first bundle, we add all our application code like components, service, modules etc. We call it as an app. We do not have to create a separate file to that. Our **main.ts** file will be the starting point for this bundle.

We put all the external libraries like Rxjs, Zone etc into a separate bundle. This includes Angular libraries also. Let us call it as the vendor. To do that we need to create the **vendor.ts** and import required libraries. Create the file called the vendor.ts under **src folder** and copy the following code



In the third bundle, we include the polyfills we require to run Angular applications in most modern browsers. Create a file called **polyfills.ts** under the **src folder** and copy the following code



### Webpack configuration

The next step is to configure the Webpack.

The Webpack by convention uses the **webpack.config.js** file to read the configuration information. Create the **webpack.config.js** in the **root folder** of our project. Add the following code



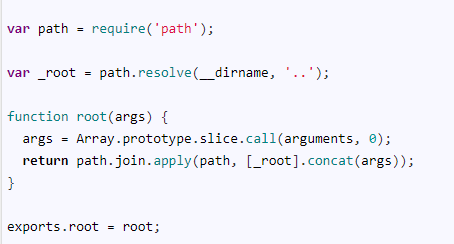
The above code tells the Webpack to read the configuration file **webpack.dev.js** from the config folder.

The Webpack can be setup so that you can have a separate configuration option for testing, development, and production. What you need to do is to create separate config files for development. testing and production and then switch between this config file in the main configuration file (webpack.config.js)

Create the folder **“config”** in the root of our project. This is where we are going to put all over Webpack related configuration option

#### Helper functions

Create the file **helpers.js** under the **folder config**and copy the following code



#### Common Configurations

Create the file **webpack.common.js** under the **folder config** and copy the following code



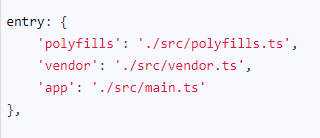




The webpack.common.js config will contain all the configuration settings, which are common to development, testing, and production builds.

Now let us look at each part





First, we let Webpack know our entry points. Remember that we have decided to create three bundles of our application. Our three entry points are **polyfills.ts** , **vendor.ts**, and **main.ts** all located in the src folder.

The Webpack starts from these files and traverses through it to find dependencies and merges all of them one bundle per each entry.

|  |
| --- |
| module: {      rules: [        {          test: /\.ts$/,          loaders: [            {              loader: 'awesome-typescript-loader',              options: { configFileName: helpers.root('src', 'tsconfig.json') }            } , 'angular2-template-loader'          ]        },        {          test: /\.html$/,          loader: 'html-loader'        },        {          test: /\.(png|jpe?g|gif|svg|woff|woff2|ttf|eot|ico)$/,          loader: 'file-loader?name=assets/[name].[hash].[ext]'        },        {          test: /\.css$/,          exclude: helpers.root('src', 'app'),          loader: ExtractTextPlugin.extract({ fallbackLoader: 'style-loader', loader: 'css-loader?sourceMap' })        },        {          test: /\.css$/,          include: helpers.root('src', 'app'),          loader: 'raw-loader'        }      ]    }, |

Webpack then uses **leaders** to transform our files. For example, the Typescript files (ts extension) are passed through “angular2-template-loade” and then to “awesome-typescript-loader” (Right to left)

|  |
| --- |
| 1  2  3  4  5 |



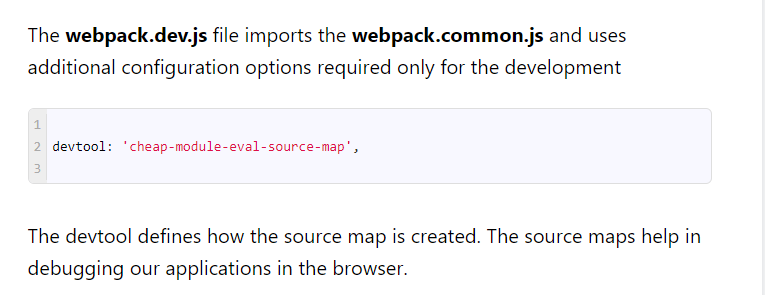
The CommonsChunkPlugin removes all the multiple used chunks of code and uses it only once.

The HtmlWebpackPlugin adds a script tag to our index.html for the each of the bundle created.

#### Development specific configuration

Create **webpack.dev.js** under the **config folder** and add the following code







Output configuration has options that affect the output of the Webpack compilation. You can configure location on disk where the compiled files are written to (path), the name of the bundle (filename), the name of the chunk file (chunkfilename) and public URL path (publicPath) etc.

You call any development environment specific plugin here. The **extract-text-**webpack**-plugin** removes the compiled CSS from the bundle and emits is as a separate file.



## Run the Application

From the command prompt type, **npm start** to run the application. Open the browser and type http://localhost:8080/. You should be able to see “Hello and welcome to Angular” message on the screen